PARALLELIZED CRC CALCULATION METHOD AND SYSTEM

FIELD OF THE INVENTION

The present invention relates generally to a parallelized CRC calculation method and system, and, more particularly, to a method and system for applications of generating and checking CRC32, especially for the Frame Check Sequence (FCS) systems.

BACKGROUND OF THE INVENTION

Cyclic redundancy code (CRC) has been used for a long time to preserve the integrity of digital data in storage and transmission systems. More particularly, CRC is an important error detection tool used for communications and data processing applications. The CRC schemes are often used for checking integrity of data because they are easy to implement and they detect a large class of errors. CRC is a kind of checksum which is transmitted with data between a source node and a target node over a communications medium. The source

node calculates the CRC for the data to be transferred using a predetermined polynomial and then transmits the data along with the CRC to the target node where the CRC of the received data is independently generated using the predetermined generator polynomial and compared with the CRC received from the source node to check if errors have occurred during the transmission. Treating the data or message as a binary polynomial, its CRC corresponding to a particular generator polynomial may be generated by raising the message polynomial to a proper power first and then taking the remainder of the message polynomial divided by the generator polynomial. For CRC generation, data bits are typically serially inputted into a CRC generator in order to produce the appropriate CRC code for transmission along with the data. Traditionally, CRC codes are generated with Linear Feedback Shift Register (LFSR) circuits. An LFSR takes the input data and shifts through a series of flip-flops on successive clock cycles. Combinations of the shift register output and data input are fed back to the flip-flops via exclusive-OR gates. An LFSR can be defined in terms of a generator polynomial which relates the input data and the CRC code via a polynomial expression and of which "+" is an exclusive-OR operation. The state of the flip-flops upon completion of the shifting process is the CRC code.

For example, ATM uses a FCS field derived from CRC

error detection codes for error checking. The integrity of the transmitted or processed message in an ATM system is ensured by the addition at the end of the message of the FCS traveling with the message itself so it can be checked on the reception side for proper transmission. The FCS code has been standardized for data integrity checking as described in the ANSI X3.139-1987 document pages 28 and 29. All the CRC codes constitute a finite Galois Field (GF), and the CRC32 codes belong to the GF generated by the following generator polynomial of degree 32:

$$g(x)=x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1.$$

This generator polynomial of degree 32 was chosen as a standard for error checking in Ethernet and then chosen by the ATM standard for AAL5 error checking. In the circuitry for calculating the FCS or checking the message, an LFSR carries out a bit by bit multiplication in the GF, i.e., modulo the polynomial on which GF is generated, and by which each bit of the message is inputted into the LFSR in the manner of most significant bit (MSB) first and division is performed by feedbacks. At the end of the process, the FCS, i.e., the remainder of the division, is within the shift registers.

Hardware implementation for CRC generators in large scale digital systems is preferred because it is faster. The

drawback of hardware implementation of CRCs is that more hardware is required with consequent increase in cost, size and complexity and a decrease in reliability. Software implemented CRCs are known although their use is not widespread because of the speed penalty thought to be inevitable. Those skilled in the art understand that choosing a polynomial of a larger degree will result in greater error detection. However, for the applications of current large scale systems, the desired hardware becomes too complicated and costly to be implemented and the required software needs huge computations. Several improvements were made for CRC generators. For example, by using CRC routines to generate tables consisting of all possible combinations of the chosen polynomial, the checksum generation is reduced to a table lookup. These CRC routines are considered to be the fastest software implementations available, but they take up a great deal of dedicated memory. Early CRC implementations use the concept of LFSR in which the polynomial division is processed one bit at a time. However, the serial processing for the generation of the CRC is relatively slow, and as the technology advanced, single-bit CRC generation was not enough to handle high-speed data processing and transmission, and parallel CRC algorithms were then developed to meet this need. The key reason that existing CRC algorithms are limited in their degree of parallelism is deeply rooted in the concept of LFSRs. All existing algorithms try to solve the same problem, i.e., how to

parallelize the bit-by-bit operation of LFSRs. As a result, the degree of parallelism never goes beyond the perceived size of LFSRs.

Accordingly, it is desired a parallelized CRC calculation method and system to reduce the processing for generation of CRC codes.

SUMMARY OF THE INVENTION

The present invention is directed to a methodology to simplify the CRC calculation, by which the process for the CRC generation is speeded up and the circuitry for the system is simplified.

To simplify the CRC calculation, according to the present invention, a linear mapping matrix corresponding to the LFSR to generate the CRC is planning and thus the computation of the LFSR to the input message for the CRC generation becomes a simplified matrix multiplication. Various linear mapping matrixes are provided for the processed message inputted in byte-wise, word-wise and doubleword-wise forms. In addition, the input messages are padded with specific dummies on the transmission side or the CRC outputs on the reception side are compared with specific patterns in accordance

with their length types for the word-wise and doubleword-wise CRC32 cases.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other objects, features and advantages of the present invention will become apparent to those skilled in the art upon consideration of the following description of the preferred embodiments of the present invention taken in conjunction with the accompanying drawings, in which:

Fig. 1 shows two schemes for a generator polynomial with message shifted into LFSR from MSB and LSB sides, respectively;

Fig. 2 shows two schemes linearly mapped from those of Fig. 1;

Fig. 3 shows the mapping matrix and its inverse for CRC32 generations;

Fig. 4 shows the circuitries for byte-wise CRC32 generations;

Fig. 5 shows the mapping matrix and its inverse for byte-wise CRC32 generations;

Fig. 6 shows the mapping matrix and its inverse for

word-wise CRC32 generations;

Fig. 7 shows the circuitries for word-wise CRC32 generations;

Fig. 8 shows the mapping matrix and its inverse for doubleword-wise CRC32 generations; and

Fig. 9 shows the circuitries for doubleword-wise CRC32 generations.

DETAILED DESCRIPTION OF THE INVENTION

Cyclic code in a systematic form

As is well-known, an *(n, k)* linear code **C** is called a cyclic code if every cyclic shift of a code vector in **C** is also a code vector in **C**. To figure out a cyclic code in a systematic form on the transmission side, let the message to be encoded is

$$M=(m_{k-1}\ldots m_1 m_0)^T, \qquad \text{(EQ-1)}$$

and the corresponding message polynomial is

$$m(x)=m_0x^{k-1}+m_1x^{k-2}+\ldots+m_{k-2}x+m_{k-1}. \quad \text{(EQ-2)}$$

After multiplying m(x) by x^{n-k}, equation EQ-2 becomes

$$x^{n-k}m(x)=m_0x^{n-1}+m_1x^{n-2}+\ldots+m_{k-2}x^{n-k+1}+m_{k-1}x^{n-k}. \quad \text{(EQ-3)}$$

Then, $x^{n-k}m(x)$ is divided by the generator polynomial g(x), and it becomes

$$x^{n-k}m(x)=q(x)g(x)+r(x). \quad \text{(EQ-4)}$$

By rearranging equation EQ-4 and inversing the sign of the remainder to replace the original one, it will be obtained the codeword polynomial

$$x^{n-k}m(x)+r(x)=q(x)g(x). \quad \text{(EQ-5)}$$

Obviously, this codeword polynomial is divisible by the generator polynomial g(x).

From the above description, it can be summarized that a cyclic encoding in a systematic form includes:

Step 1. Multiplying the message m(x) by x^{n-k};

Step 2. Deriving the remainder r(x) by dividing $x^{n-k}m(x)$ by the

generator polynomial g(x); and

Step 3. Combining r(x) with $x^{n-k}m(x)$ to obtain the codeword polynomial $x^{n-k}m(x)+r(x)$.

Likewise, in order to check the integrity of the received codeword on the reception side, it is verified if the received sequence is divisible by the generator polynomial g(x).

Shortened cyclic codes

Given an *(n, k)* cyclic code **C**, if the set of the code vectors for which the *l* leading high-order information digits are identical to zero, then there are 2^{k-l} such code vectors and they form a linear subcode of **C**. If the *l* zero information digits are deleted, it is obtained a set of 2^{k-l} vectors of length n-*l*. These shortened vectors form an *(n-l, k-l)* linear code, and which code is called a shortened cyclic code and is not cyclic.

Implementation of divisor

No matter for a cyclic code encoding or decoding, a divisor of Galois Field GF(2) is needed. For example, a simple Linear Feedback Shift Register (LFSR) is employed to implement the divisor. Furthermore, depending on the dividend sequence shifted into LFSR either from MSB side or Least Significant Bit

(LSB) side, there are two schemes for implementation of a divisor, i.e.,

Scheme 1: Message is shifted into LFSR from MSB side, which is mathematically equivalent to

$$m(x)x^{n-k} \bmod g(x) \qquad \text{(EQ-6)}$$

Scheme 2: Message is shifted into LFSR from LSB side, which is mathematically equivalent to

$$m(x) \bmod g(x) \qquad \text{(EQ-7)}$$

For illustration, two circuitries are shown in Fig. 1 for these two schemes for the generator polynomial $g(x)=x^3+x^2+1$.

Linear mapping

Further, the linear feedback shift registers shown in Fig. 1 can be regarded as a linear mapping mathematically, as shown in Fig. 2. For the same generator polynomial $g(x)=x^3+x^2+1$, it can be derived the G mapping:

$$g_0(2)=g_i(2) \oplus g_i(1), \qquad \text{(EQ-8a)}$$

$$g_0(1)=g_i(0), \text{ and} \qquad \text{(EQ-8b)}$$

$$g_o(0)=g_i(2), \qquad \text{(EQ-8c)}$$

and this linear mapping can be represented in a matrix form as

$$\begin{bmatrix} g_o(2) \\ g_o(1) \\ g_o(0) \end{bmatrix} = G \begin{bmatrix} g_i(2) \\ g_i(1) \\ g_i(0) \end{bmatrix} \qquad \text{(EQ-9)}$$

where

$$G = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \qquad \text{(EQ-10)}$$

and trivially, the matrix G is invertible and its inverse matrix is

$$G^{-1} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \qquad \text{(EQ-11)}$$

Based on the Scheme 1 and 2, there exist recursive equations between the output of the D-type flip-flops of the polynomial generator g(x) and the input of the encoded message, respectively, as

Scheme 1:

$$R(k)=G(R(k-1)+M(k-1)), \text{ and} \qquad \text{(EQ-12)}$$

Scheme 2:

$$R(k)=GR(k-1)+M(k-1). \qquad \text{(EQ-13)}$$

Further tracing the output of the D-type flip-flops, i.e., the remainder of a division, in Scheme 1, it results in

$$R(0)=I, \qquad \text{(EQ-14a)}$$

$$R(1)=G(R(0)+M(0))=GI+GM, \qquad \text{(EQ-14b)}$$

$$R(2)=G(R(1)+M(1))$$

$$=G^2I+G^2M(0)+GM(1), \qquad \text{(EQ-14c)}$$

...

$$R(k)=G(R(k-1)+M(k-1))$$

$$=G^kI+G^kM(0)+G^{k-1}M(1)+\ldots+GM(k-1) \qquad \text{(EQ-14d)}$$

Generation of FCS

In Standard 802.3, the CRC32 is employed to generate FCS and the generator polynomial is

$$g(x)=x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+$$

$$x^5+x^4+x^2+x+1. \qquad \text{(EQ-15)}$$

Mathematically, the CRC value corresponding to a given frame is defined by the following procedures:

a.) The first 32 bits of the frame are complemented;

b.) The k bits of the frame are then considered to be the coefficients of a polynomial m(x) of degree k-1;

c.) m(x) is multiplied by x^{32} and divided by g(x), producing a remainder r(x) of degree less than or equal to 31;

d.) The coefficients of r(x) are considered to be a 32-bit sequence; and

e.) The bit sequence is complemented and the result is the FCS f(x).

In the procedure a, disclosed are two implementation methods:

Method 1: complementing the first 32bits of the message directly; and

Method 2: initiating the D-type flip-flop with 1 specific value, e.g., 0xffffffff for the Scheme 1 and 0x46af6449 for the Scheme 2.

The mapping matrix G and its inverse matrix G^{-1} are shown in Fig. 3.

On the reception side, when the whole of frame is acquired, the output of the Scheme 1 CRC checker is compared with the value of 0xc704dd7b to examine the integrity of the received frame. The reason is explained herewith.

Let the transmitted message (except for FCS) is represented in a polynomial form

$$m(x)=m_0x^{k-1}+m_1x^{k-2}+\ldots+m_{k-2}x+m_{k-1}, \quad \text{(EQ-16)}$$

and defining a polynomial c(x) of degree 31 with all of its coefficients to be

$$c(x)=1x^{31}+1x^{30}+\ldots+1x^2+1x+1, \quad \text{(EQ-17)}$$

then the remainder r(x) generated by the procedure c will be

$$r(x)=(m(x)+c(x)x^{k-32})x^{32}\text{mod}g(x). \quad \text{(EQ-18)}$$

After performing procedures d to e, it will generate FCS and the transmitted sequences

$$f(x)=\overline{r(x)}, \text{ and} \quad \text{(EQ-19)}$$

$$n(x)=m(x)x^{32}+f(x) \quad \text{(EQ-20)}$$

On the reception side, if the integrity of this frame sequence is maintained, then the remainder of a division will be

$$s(x)=(n(x)+c(x)x^k)x^{32}\bmod g(x)$$
$$=(m(x)x^{32}+f(x)+c(x)x^k)x^{32}\bmod g(x)$$
$$=(m(x)x^{32}+c(x)x^k+f(x))x^{32}\bmod g(x) \quad \text{(EQ-21)}$$

From equation EQ-18, equation EQ-21 can be further modified to be

$$s(x)=(q(x)g(x)+r(x)+f(x))x^{32}\bmod g(x)$$
$$=(r(x)+f(x))x^{32}\bmod g(x)$$
$$=c(x)x^{32}\bmod g(x)$$
$$=[0xc704dd7b][x^{31}\ldots x^1\ 1] \quad \text{(EQ-22)}$$

Based on a similar derivation, it can be further obtained, if the Scheme 2 is adopted, that the checking pattern will be the value of 0xffffffff.

Parallelized CRC calculation

So far the encoding message is sequentially inputted to the CRC calculation with one bit each time, however, for high-speed applications, CRC calculation is desired for the

capability of multiple message bits inputted, e.g., byte-wise, at a time to increase the throughput. Consequently, the principal architecture of the previous proposed two schemes is maintained and there is only somewhat difference at the mapping matrix.

Let the input message and the status of the flip-flops be represented, respectively, with a vector form as

$$M(k)=[m_k\ 0\ \ldots\ 0]^T, \text{ and} \quad \text{(EQ-23)}$$

$$R(k)=[r_k^{31}\ r_k^{30}\ \ldots r_k^{0}]^T. \quad \text{(EQ-24)}$$

Tracing the R(k) influenced by the values of M(k),

$$\text{initially,}\quad R(0)=0, \quad \text{(EQ-25a)}$$

$$\text{then,}\quad R(1)=G(R(0)+M(0))=GM(0), \text{ and} \quad \text{(EQ-25b)}$$

$$R(2)=G(R(1)+M(1))=G^2M(0)+GM(1). \quad \text{(EQ-25c)}$$

It can be further verified

$$G^2M(0)=m_0\times\text{the 1st column of the } G^2 \text{ matrix, and} \quad \text{(EQ-26a)}$$

$$GM(1)=m_1\times\text{the 1st column of the G matrix.} \quad \text{(EQ-26b)}$$

Defining a new vector with l ($\leqq 32$) non-zero entries as

$$M_l(k)=[m_k^{l-1}\ m_k^{l-2}\ \ldots m_k^0\ 0\ \ldots 0]^T \quad \text{(EQ-27a)}$$

$$=[m_{k*1}\ m_{k*1+1}\ \ldots m_{k*1+l-1}\ 0\ \ldots 0]^T \quad \text{(EQ-27b)}$$

When l=2 and k=0, it becomes

$$M_2(0)=[m_0\ m_1\ \ldots 0\ \ldots 0]^T, \quad \text{(EQ-28)}$$

and it is further derived

$G^2M_2(0)=m_0\times$the 1st column of the G^2 matrix +
$m_1\times$the 2nd column of the G^2 matrix. (EQ-29)

Examining the property of the G matrix, it can be found

the 1st column of the G matrix =
the 2nd column of the G^2 matrix. (EQ-30)

Hence, it is obtained

$$G^2M_2(0)=G^2M(0)+GM(1), \quad \text{(EQ-31)}$$

and it is further included that

$$G^lM_l(0)=G^lM(0)+G^{l-1}M(1)+\ldots+GM(l-1), \quad \text{for} \quad l \leqq 32. \quad \text{(EQ-32)}$$

Byte-wise CRC32

When the message is inputted in byte-wise form at a time, the input message and calculated remainder vectors are represented as

$$M_8(k) = [m_k^7 \ m_k^6 \ \dots \ m_k^0 \ 0 \dots 0]^T_{1\times 32} \quad \text{(EQ-33)}$$

$$R(k) = [r_k^{31} \ r_k^{30} \ \dots \ r_k^0]^T_{1\times 32} \quad \text{(EQ-34)}$$

If the Scheme 1 is adopted, the recursive equation of the input message and calculated remainder is

$$R(k+1) = T(R(k)+M_8(k)), \quad \text{(EQ-35)}$$

and the circuitry is shown in Fig. 4.

Likewise, if the Scheme 2 is adopted, the recursive equation will be

$$R(k+1) = TR(k)+M_8(k), \quad \text{(EQ-36)}$$

and its circuitry is also shown in Fig. 4.

For equation EQ-35 and 36, the mapping matrix T and

its inverse are shown in Fig. 5, and of which, the number on the right-hand side of each row indicates how many nonzero entries that row has. For example, in the matrix T, Row 1 has 4 nonzero entries and those rows with maximum nonzero entries, the value of 7, are Row 5, 12 and 13.

Word-wise CRC32

When the message is inputted in word-wise form at a time, the input message and calculated remainder vectors are

$$M_{16}(k) = [m_k^{15}\ m_k^{14}\ \dots\ m_k^{0}\ 0\ \dots\ 0]^T_{1\times 32} \quad \text{(EQ-37)}$$

$$R(k) = [r_k^{31}\ r_k^{30}\ \dots\ r_k^{0}]^T_{1\times 32} \quad \text{(EQ-38)}$$

Similar to the situations for the byte-wise form, the recursive equations for the Scheme 1 and 2 are

$$R(k+1) = T(R(k)+M_{16}(k)) \text{ for Scheme 1, and} \quad \text{(EQ-39)}$$

$$R(k+1) = TR(k)+M_{16}(k) \text{ for Scheme 2,} \quad \text{(EQ-40)}$$

and for which, the matrix T and its inverse T^{-1} are shown in Fig. 6, and their circuitries are shown in Fig. 7.

However, the MAC frame is based on octet format, and the length of the processed message is not always divisible by 2.

As a result, some dummies are padded on the message in order to have the word-wise format when the word-wise CRC calculation is employed. Two strategies for dummy padding are further proposed:

Strategy 1: padding with some zero-valued octets before the prefix of the processed message for the transmission side; and

Strategy 2: padding with some zero-valued octets after the suffix of the processed message for the reception side.

In the word-wise case, no matter what the length of a frame is, they can be classified in accordance with their length into two types: 2n and 2n+1. If the Strategy 1 is adopted, the initial values of the flip-flops will vary with the length type as listed in

Table 1

Length type	Padding number	C(0)	C(1)	C(2)
2n+1	1	00 ff 00 00	ff ff 00 00	ff 00 00 00
2n	2	00 00 00 00	ff ff 00 00	ff ff 00 00

In order to complement the first 32 bits of the processed message, C(0), C(1) and C(2) vectors in Table 1 are added to the first 3 message blocks, and the influences of the C(i) on the

calculated R(x) follow the relations

$$R(1)=TC(0) \quad \text{(EQ-41a)}$$

$$R(2)=T(R(1)+C(1))=T^2\,C(0)+TC(1), \text{ and} \quad \text{(EQ-41b)}$$

$$R(3)=T(R(2)+C(2))=T^3\,C(0)T^2\,C(1)+T\,C(2), \quad \text{(EQ-41c)}$$

and the initial value of R(x) after 3 times of iterations is equivalent to the influences of C(i) on it, i.e.,

$$T^3\,R(0)= T^3\,C(0)+T^2\,C(1)+T\,C(2), \text{ or} \quad \text{(EQ-42a)}$$

$$R(0)= C(0)+T^{-1}\,C(1)+T^{-2}\,C(2), \quad \text{(EQ-42b)}$$

and summarized in

Table 2

Length type	The initial value of R(0)
2n+1	0x9bf1a90f
2n	0x09b93859

For the Strategy 2, the resultant output of the flip-flops will vary with the length of the processed frame, and which implies, for examining the integrity of a received frame, that the output Pi will be compared with a specified pattern depending on the length type i as

$$P_i = G^{8i}\,[0xc704dd7b]^T, \text{ for } i = 1 \text{ and } 2, \quad \text{(EQ-43)}$$

and listed in

Table 3

Length type	Padding number	Pattern (P_i)
2n+1	1	0x4710bb9c
2n	2	0x3a7abc72

Doubleword-wise CRC 32

When the message is inputted in doubleword-wise form at a time, the input message and calculated remainder vectors are

$$M_{32}(k)=[m_k^{31}\ m_k^{30}\ \ldots\ m_k^{0}]_{1\times 32}^{T}, \text{ and} \qquad \text{(EQ-44a)}$$

$$R(k)=[r_k^{31}\ r_k^{30}\ \ldots\ r_k^{0}]_{1\times 32}^{T}. \qquad \text{(EQ-44b)}$$

Similar to the byte-wise case, the recursive equation R(k) for the Scheme 1 and 2 are

$$R(k+1) = T(R(k)+M_{32}(k)), \text{ and} \qquad \text{〔EQ-45a〕}$$

$$R(k+1) = T\,R(k)+M_{32}(k), \qquad \text{〔EQ-45b〕}$$

where the matrix T is defined in Fig. 8, and the corresponding circuitries are shown in Fig. 9.

For the initial value of the flip-flops in the Scheme 2 in the above description, it can be derived by

$$G^{32}R(0) = [0xffffffff]^T \quad \text{(EQ-46a)}$$

$$R(0) = (G^{32})^{-1}[0xffffffff]^T$$

$$= [0x46af6449]^T \quad \text{(EQ-46b)}$$

Again, no matter what the length of a frame is, they can be classified into 4 types: 4n, 4n+1, 4n+2 and 4n+3.

When the Strategy 1 is adopted, the initial values of the flip-flops will vary with the length type as denoted in

Table 4

Length type	Padding number	C(0)	C(1)
4n+3	1	00 ff ff ff	ff 00 00 00
4n+2	2	00 00 ff ff	ff ff 00 00
4n+1	3	00 00 00 ff	ff ff ff 00
4n	4	00 00 00 00	ff ff ff ff

The recursive equations are

$$R(1) = TC(0), \text{ and} \quad \text{(EQ-47a)}$$

$$R(2) = T(R(1)+C(1)) = T^2C(0)+TC(1) \quad \text{(EQ-47b)}$$

Let the initial values of the flip-flops be R(0), so that

$$T^2R(0) = T^2C(0) + TC(1), \text{ or} \quad \text{(EQ-48a)}$$

$$R(0) = C(0) + T^{-1}C(1), \quad \text{(EQ-48b)}$$

and R(0) is listed in

Table 5

Length type	The initial value of R(0)
4n+3	0x9bf1a90f
4n+2	0x09b93859
4n+1	0x816474c5
4n	0x46af6449

In the strategy 2, the resultant output of the flip-flops will vary with the length of the processed frame, which result implies, for examining the integrity of a received frame, the output Pi is compared with a specified pattern depending on the length type i, in the following rule

$$Pi = G^{8i}[0xc704dd7b]^T, \text{ for } i = 1, 2, 3 \text{ and } 4, \quad \text{(EQ-49)}$$

and the pattern in

Table 6

Length type	Padding number	Pattern (Pi)
4n+3	1	0x4710bb9c
4n+2	2	0x3a7abc72
4n+1	3	0x8104c946
4n	4	0x69044bb59

While the present invention has been described in conjunction with preferred embodiments thereof, it is evident that many alternatives, modifications and variations will be apparent to those skilled in the art. Accordingly, it is intended to embrace all such alternatives, modifications and variations that fall within the spirit and scope thereof as set forth in the appended claims.